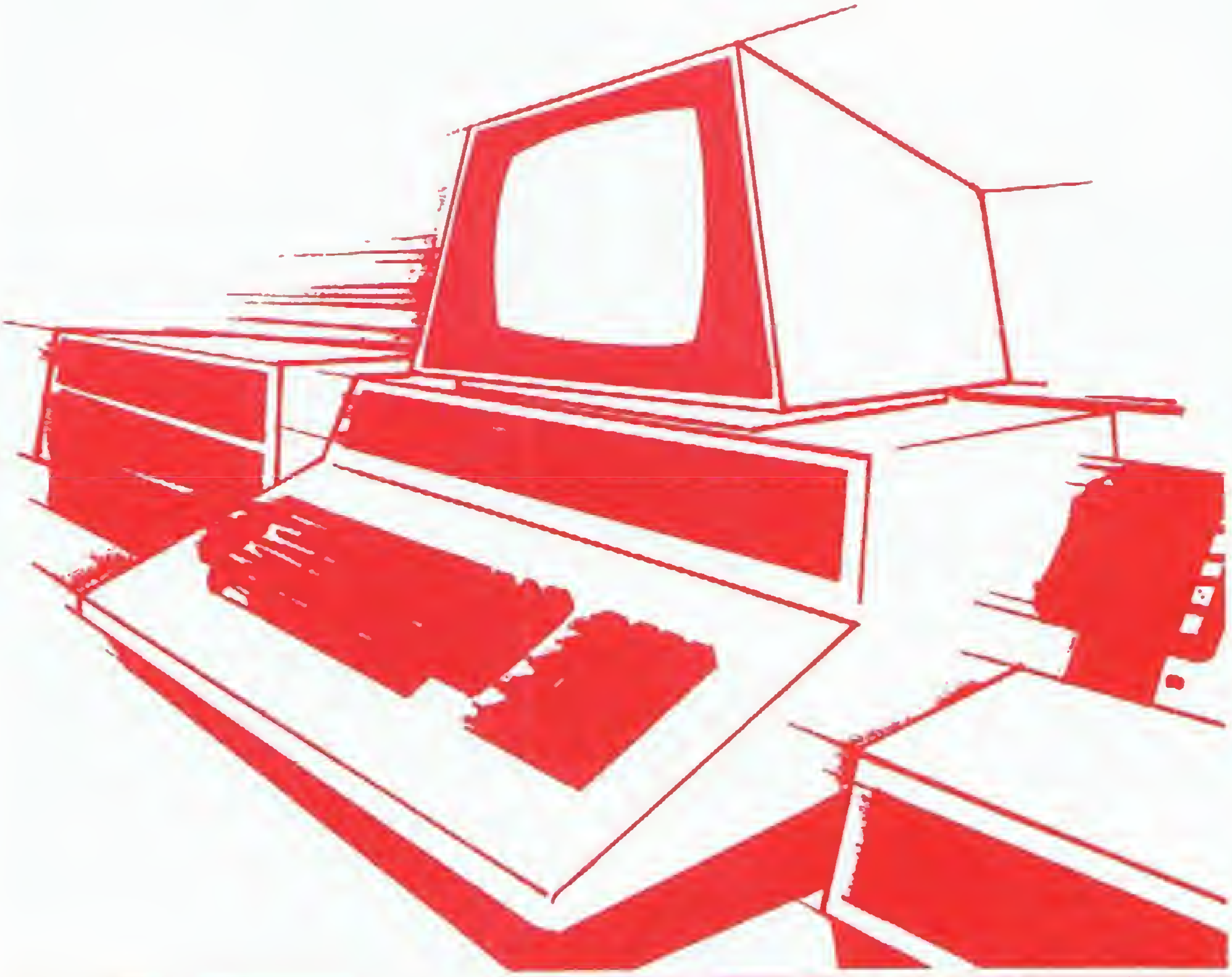
CPUCN

The Official Commodore Pet Users Club Newsletter



Volume 2

Issue 4

commodore

Contents

We would like to thank John Kyle Price and the Bristol Software Factory for allowing us to reproduce their artwork on the cover of CPUCN.

Petpacks

Pete Gerrard

Last issue featured an extensive review of 'Invaders', MP067, which has since proved to be one of our most popular sellers.

'Invaders' provided the spark of inspiration for the ARCADE series, the aim being to produce arcade style games of exceptional quality, at a reasonable price and presented in an appealing package. So far there are three games in the series, and the two new ones are 'Acrobat', MP068, and '3D Startrek', MP069. Both of these are excellent value at £7.

'Acrobat', like 'Invaders', is a game already well known in amusement arcades and public houses. A description of the game may seem slightly "ridiculous" to those of you who don't know it ; to those who do, bear with me.

You control a see-saw, and try to bounce two little men up and down from it and make them hit any of three rows of balloons which are moving across the top of the screen. When the man comes down again, you move the see-saw along so that he can bounce back up again. If you miss him, he crumples to the floor and is carted off in an ambulance. If you connect your Pet up for sound, you'll hear some marvellous effects here, and more tunes are played if you manage to get a bonus by clearing one complete row of balloons.

At the start of the game the see-saw is placed to one side of the screen, with one man on it, and the other man appears from a spring board half way up the screen. Those spring boards can get in the way!

Altogether you have five lives, or six if you manage to clear the top row of balloons. Movement of the see-saw is very easy ; just press 4 to move left and 6 to move right. Once you've got the hang of things the game becomes extremely addictive, and as the majority of the game is written in machine code the action can get very fast, and the graphics are quite superb. Definitely a game you won't be able to switch off!

3D Startrek is one game you won't have seen before, as it's been written just for the Pet. Since it is, in the opinion of many people, the best games program ever written for the PET (and one of the best programs full stop!) it just had to be included in the Arcade series. Complete with a five page instruction manual this totally machine code program uses every available byte on an 8K Pet, and is packed with exciting features. When demonstrated at a recent software exhibition the comments ranged from 'fantastic' to 'incredible' to the unprintable from one person who couldn't believe what he was seeing!

So what's all the fuss about? It's only another Startrek game. But what a game! The standard display is a short range scan of the sector you are currently in, showing the position of yourself, stars, enemy vessels and bases (if there are any of course), and your own bases if they're there, and you have two of those. All this is in a continuous, real-time, state of movement.

The two other displays that can be instantly called up, and which remain as long as the relevant key is held down, are a damage report of how you've been doing in battle, showing how badly anything might be damaged and consequently how long it will take to repair, and a long range scan of the whole universe, showing the whereabouts of the enemy and yourself.

Movement can be done in two fashions, impulse or warp drive. Impulse drive is a speed from 1 to 9, and you control your movement by the cursor control keys, which turn you to port and starboard. Warp drive is again a speed from 1 to 9, and for the first time you get some indication of the power of the program when you see yourself travelling through space in a three dimensional display of stars whizzing past you as the power builds up.

To engage the enemy in battle, which after all is the purpose of your mission, you have to approach their vessels, being careful to dodge the photon torpedoes being fired at you, and perhaps launching a few of your own. If this is survived and you manage to lock onto battle stations, you'll see a display so stunning you'll probably be hit before you can fire back! The screen has become your window into space, and there's the enemy vessel dodging about as you try and manoeuvre to get him in the centre of your sights. You have full control of movement, and can watch the stars scrolling in every direction while you try and line the enemy up for the kill. Once into battle stations it's you or him - there's no retreating!

That then is a game that is so well written you'll be playing it again and again as you try and save the universe from disaster. Quite simply the best game ever written for the PET, and at only £7 as well!!

Upper/Lower Case Converter

A couple of bugs crept into the upper-lower case convertor published in CPUCN Volume 2 Issue 3. Here is an easy to use version which has been de-bugged:-

```
10 PRINT"       UPPER/LOWER CASE CONVERTER"
20 PRINT
30 PRINT"THIS MACHINE CODE PROGRAMME CONVERTS"
40 PRINT"THE SCREEN TEXT OF A PROGRAMME WRITTEN"
50 PRINT"FOR THE OLD PET INTO THE NEW VERSION"
60 PRINT"AND VICE VERSA. THE BASIC PROGRAMME"
70 PRINT"LINES ARE CHANGED ALSO SO THAT ALL YOU"
80 PRINT"HAVE TO DO IS TO LOAD THE PROGRAMME TO"
90 PRINT"BE ALTERED NOW. TYPE 'SYS826','RETURN'"
100 PRINT"AND RE-RECORD THE AMENDED VERSION."
110 FORJ=826TO906:READA
120 POKEJ,A:NEXT
130 DATA169,4,133,202,169,1,133,201
140 DATA32,89,3,160,0,196,202,240,13
150 DATA177,201,170,200,177,201,134
160 DATA201,133,202,76,66,3,96,160,4
170 DATA177,201,240,249,201,34,240,4
180 DATA200,76,91,3,200,177,201,240
190 DATA236,201,34,240,23,201,65,144
200 DATA243,201,91,144,4,201,192,144
210 DATA235,201,219,176,231,73,128
220 DATA145,201,76,103,3,200,76,91,3
230 NEW
READY.
```

```
33A   A9 04         LDA   #$04       ;)E
33C   85 CA         STA   $CA        ;FJ
33E   A9 01         LDA   #$01       ;)B
340   85 C9         STA   $C9        ;FI
342   20 59 03      JSR   $0359      ; YD
345   A0 00         LDY   #$00       ; A
347   C4 CA         CPY   $CA        ;DJ
349   F0 0D         BEQ   $358       ;0N
34B   B1 C9         LDA   ($C9),Y    ;1I
34D   AA            TAX              ;*
34E   C8            INY              ;H
34F   B1 C9         LDA   ($C9),Y    ;1I
351   86 C9         STX   $C9        ;GI
353   85 CA         STA   $CA        ;FJ
355   4C 42 03      JMP   $0342      ;LBD
* * * * * * * * * * * * * *
358   60            RTS              ;
* * * * * * * * * * * * * *
359   A0 04         LDY   #$04       ; E
35B   B1 C9         LDA   ($C9),Y    ;1I
35D   F0 F9         BEQ   $358       ;09
35F   C9 22         CMP   #$22       ;I"
361   F0 04         BEQ   $367       ;0E
363   C8            INY              ;H
364   4C 5B 03      JMP   $035B      ;L[D
* * * * * * * * * * * * * *
367   C8            INY              ;H
368   B1 C9         LDA   ($C9),Y    ;1I
36A   F0 EC         BEQ   $358       ;0,
36C   C9 22         CMP   #$22       ;I"
36E   F0 17         BEQ   $387       ;0X
370   C9 41         CMP   #$41       ;IA
372   90 F3         BCC   $367       ;Q3
374   C9 5B         CMP   #$5B       ;I[
376   90 04         BCC   $37C       ;QE
378   C9 C0         CMP   #$C0       ;I@
37A   90 EB         BCC   $367       ;Q+
37C   C9 DB         CMP   #$DB       ;I[
37E   B0 E7         BCS   $367       ;0'
380   49 80         EOR   #$80       ;IA
382   91 C9         STA   ($C9),Y    ;RI
384   4C 67 03      JMP   $0367      ;L'D
* * * * * * * * * * * * * *
387   C8            INY              ;H
388   4C 5B 03      JMP   $035B      ;L[D
* * * * * * * * * * * * * *
```

Direct Access

MIKE GROSS-NIKLAUS

1. AIM

While the Disk Operating System for the CBM 3040 disk drive supports sequential files to a greater extent than it does direct access files, all the tools for handling direct access are present. Provided you understand what they are and what they can do, they are quite simple to use and flexible enough to build up sophisticated direct access systems.

This article explains the various components of a direct access system, how to link them together and how to code them into your programs. The system described is simple, in fact it is the one used as an example on our disk utilisation course. Once you understand it, the way is open for you to design and implement your own made-to-measure direct access disk routines.

2. THE WAY INFORMATION IS ORGANISED ON THE DISKETTES

The information recorded on the diskette is organised into a series of tracks and, within each track, into compartments called sectors. There are 35 tracks on each diskette. The number of sectors per track varies. The outside tracks, the longest, have 21 sectors per track and the inside ones, the shortest, only 17 sectors. The total number of sectors on the diskette is 690. An alternative name for a sector is a block. Conventionally one refers to information being contained in track 2, sector 3 for example or in block 25. Each block contains space for 255 characters.

A collection of items of information is called a file. Let's suppose you have a file containing the surname, first name and telephone number of all your business contacts and that these are held in alphabetical order. The details for each person in the file are collectively called a record. The pieces of information within each record are oalled items. In our example there are three items in each record.

3. SEQUENTIAL FILES

There are two main ways you can record this information onto the disk and retrieve it. One is called sequential and the other direct access. Each method has its advantages and disadvantages. Part of the task of planning a disk based program or programs suite is to decide which method to use for holding particular files.

Let us consider sequential files first. In sequential recording, the Disk Operating System looks after the organisation of information on the disk into tracks and sectors. You tell the DOS what the next item of information to be recorded is and the DOS records it starting at the 'next available' character position on the disk, stradling two blocks if necessary, and avoiding blocks already in use by other files. Diagram 2 illustrates the concept.

One advantage of sequential organisation is that where your items vary in length, becuase they are packed one after the other on the diskette, no space is wasted.

A disadvantage of sequential files is that no record is kept to show you at what character in which sector of a particular track a specified item begins. Thus to read a particular item from a sequential file on the disk, it is necessary to read everything in the file which was recorded ahead of the required item, either counting items or looking for information which will uniquely identify it.

Using our example file, the program will have to search perhaps through many records before finding the details for ZAKS, RODNEY. Moreover, supposing you then want the phone number for SPENCER C, the programme will have to start over from the beginning of the file and make another long search. Also, the only way to amend an item in a sequential file is to write an updated version of the entire file.

Don't get the idea that sequential files are useless or second best, however. There are many situations where they are ideal. For example, when processing a payroll, where personal details and brought forward totals are matched against ourrent information on hours worked and rates, both brought forward, current and carry forward files are ideally held in sequential format.

4. DIRECT ACCESS FILES

The other mode of operation is called direct access. It allows you to specify into which sector on which track and starting at which character a particular item of information is to go. And provided you keep a note of where you put the information, you can directly access it at a later time, without having to wade through all the other records. Because this allows you to examine items in the file in a non-sequential, unpredictable order, direct access files are often termed random access files. Since you can both read from and write to a particular place on the diskette, you can amend items in direct access files without rewriting the whole file.

Currently the disadvantage of direct access is that DOS doesn't help you a great deal. You must keep track of where you put the information. The direct access file is not listed in the diskette directory, and indeed no name for it is required by DOS. If you put your direct access file on a diskette containing programme or sequential files, you must take precautions to ensure you don't write into blocks in use by those files.

It is normal when using direct access to make the items fixed length. In our example, we might specify the surname as a key word of 20 characters, the first name as 10 characters and the telephone number as 20. In use, MIKE and ALFRED will both occupy 10 characters 'slots' in their respective records. As a result, direct access files use more space than sequential access files to hold the same information.

5. THE COMPONENTS OF A DIRECT ACCESS SYSTEM

When building a direct access system,you must decide how long each fixed length item is to be. In our example, the items are 10 and 20 characters long. The total number of characters per record is 30, so it would be possible to fit eight records into each block. However, to keep things simple while you learn the techniques, the example will use 1 record per block only.

The components required to write one record per block are:-

a. At the start of direct access operations, open up a route from PET to a buffer in the disk unit.
b. Copy the record from one or more BASIC variables into that buffer, starting at character position 1.
c. Find the next available block on the diskette.
d. Tell the DOS that the block is reserved.
e. Write the whole buffer to that block.
f. Make an entry in an index array relating the block to the record key. The record key is the index word you use to look up the record details. In our example it will be the surnames of the various people in the telephone file.
g. At some stage thereafter, save the index array as a sequential file.

Once the index has been saved, the system can be closed down, switched off even, and later opened up again.

To read a record from the disk requires the following components:-

h. Read the index back from the sequential file into a BASIC array.
i. At the start of direct access operations open up a route from a disk buffer to the PET.
j. Search the index for the keyword of a required record and note the track and sector number associated with it.
k. Read the whole of the sector specified in the index from the diskette into the DOS buffer reserve.
l. Copy the information from the buffer into one or more BASIC variables.

Finally, to amend a direct access record:-

m. Read the whole block into a disk buffer as shown in h-k above.
n. Point at the part of the buffer to be overwrittten.
o. Copy the new information into the buffer from one or more BASIC variables over-writing the specified portion of the information in the buffer.
p. Write the contents of the buffer back to the block it came from.

6. WRITING A DIRECT ACCESS RECORD

Opening up the command route

The disk unit is normally set up as device number 8 on the IEEE 488 information bus. It has built-in software which comes into operation when the device is attached to the PET and both are switched on. Associated with this software are several information routes, or channels, within the disk unit. Of particular interest to us here are channels 2 to 14 which are routes for information to be written onto or read from the disk, and channel 15 which is the route for commands to the disk software and any messages from it.

It is good practice to open the command channel near the beginning of the program and to close it just before the program ends. The syntax is OPEN logical file number,8,15. The logical file number is used by BASIC as a key to the other details to save you having to type them in each time. It can be any number from 1 to 255 but is commonly made the same as the channel number. Thus OPEN15,8,15

Opening up a data route

DOS allows 5 disk data routes plus the command route to be open at one time. A data route can either be opened at the beginning of the program in the same way as the command channel, or if many files are opened in the program, opened and closed around each access or set of access calls. However, closing the file causes a number of time-consuming (up to 2 seconds total) processes to take place within DOS. It is better to leave the route open if you can. You need to reserve a buffer to hold the information going to or coming from the diskette, and this is specified in the OPEN command. The syntax is OPEN logical file,8, channel,"#" where the '#' reserves the next available buffer and associates it with the channel. Suppose we use channel 2 then the statement would be:- OPEN2,8,2,"#"

Copying the data from a BASIC variable

Once the route is opened, information can be copied from a variable to the buffer associated with the data channel using PRINT# logical file number, variable name. For example, supposing the first name and phone number had been padded to 10 and 20 characters respectively and then

concatenated into a 30 character string. (The keyword need not be part of the direct access record). e.g.

```
MIKE.....01 388 5702.......
```

If the formatted information were in A$, then it could be transferred to the buffer by PRINT#2,A$ assuming a data route had been opened as logical file 2.

In order to allow specific parts of the buffer to be overwritten, (see 5 m-p above), the information is written into the buffer starting at the character pointed at by a buffer pointer held within DOS. To alter the position of this pointer, we must command DOS using channel 15. All the DOS commands can be shortened to their initial letters. The syntax for moving the block pointer is, (where logical file 15 has been opened as the command channel):-

```
PRINT#15,"B-P";C;P
```

where C is the data channel number and P is the pointer position required. So a sequence for getting our telephone record into the buffer would be:-

```
PRINT#15,"B-P";2;1:REM BLOCK POINTER
COMMAND TO DOS

PRINT#2,A$         :REM DATA
```

Finding a free block

A simple way of finding spare blocks is to put the direct access files onto an otherwise empty diskette. Then, provided you don't use track 18, sectors 0 to 2, which are reserved for the disk directory, you control completely the placing of information on the diskette. If you want to use a diskette which contains or will contain files placed there by DOS, then you will need to find out which blocks are free and reserve them. This is acheived in a rather indirect way, using the Block Allocate command.

When you use the Block Allocate command to tell DOS that a block is reserved, DOS will reply with a message which you can read from the command channel. If the block was free to be allocated, DOS will allocate it and reply with message 0, "OK". If that block was already in use, DOS replies with message 65, "NO BLOCK", and the track and sector number of the next free block 'down the disk'. So you can always find the next free block by attempting to allocate track 1, sector 0. The very first time that is done, it will be allocated, and from then on, the command channel will tell you where the next free block is. Having found it, we can allocate it, (which should give an OK message) and then write into it. From then on, when DOS writes its sequential or program files, it will avoid that block. The syntax to allocate a block requires the drive as well as the track and sector. Thus:-

```
PRINT#15,"B-A";0;1;0
```

attempts to allocate drive 0, track 1, sector 0.

```
INPUT#15,EN,EM$,ET,ES
```

reads the resultant message from the command channel. If EN = 0 then the block will be allocated. If EN = 65 then:-

```
PRINT#15,"B-A";0;ET;ES
```

will allocate the next free block which is track ET, sector ES.

Writing the contents of the buffer to disk

Again you use a DOS command, Block Write. The channel (and hence the associated buffer), the drive, the track and the sector must all be specified. The track and sector you already know. They are ET and ES. So:-

```
PRINT#15,"B-W";2;0;ET;ES
```

will write the contents of the buffer associated with channel 2 to track ET, sector ES on drive 0.

Handling the index

The index associates the key-word with the track and sector number of the information you put on the disk. The best way of holding it is in array form. Since the track and sector numbers are numeric and the keyword is alpha, you can use two 'parallel' arrays, one string and one numeric or you could hold all three items in a three wide string array and convert the track and sector numbers from and to numeric form using STR$ and VAL. In our example, supposing there were just three entries, held in the three columns of I$(,), they would look like this:-

GROSS-NIKLAUS	1	0
SPENCER	1	1
ZAKS	1	2

If the next record added were that for GOLTZ, row 4 of the array might be:-

GOLTZ	1	3

Once the index gets large, searching from the beginning to find the required key word takes time, so at some stage, this array should be sorted into order, or you could arrange for each new item to be inserted into the correct position as it is entered. Once keys are in alphabetic order, it is possible to do a binary search to shorten the time to find one particular item. A binary search is similar to the 'Guess a number game' where one player (or the PET!) hides a number between one and a hundred, and you must guess it in as few turns as possible. In a binary search you would guess 50 to start with, then 25 or 75 depending on whether your guess was high or low. Similarly with an alphabetic binary search. You compare the keyword with the

element halfway down the array. If the element is 'higher' than the keyword then you halve the gap and look at the element a quarter down the array, and so on until you find a match or until the next 'move' is less than 1, in which case the keyword is not in the index.

Reading data back again

First, get the keyword. In our example it might come from the keyboard using an INPUT statement. Then the index is searched, using a binary chop perhaps if the index is likely to be a long one, say over 200 entries. Once the entry for the keyword is found, the track and sector number of the details are available.

The entire block is read into a buffer using "U1" in preference to "B-R" which tackles matters in a way inappropriate to this example. The same channel as that used for writing can be used. Thus:-

```
PRINT15,"U1";2;0;T;S
```

will read the information from track T, sector S of drive 0 into the buffer associated with channel 2.

The block pointer is used to specify from where in the buffer INPUT is to start. In our case it is character position 1, so:-

```
PRINT#15,"B-P"2;1
INPUT#2,A$
```

will read our concatenated details string into A$.

Amending a direct access record

Remember the procedure is to read the block into the buffer, overwrite the required chunk of the buffer then write the block back to the disk.

Assuming you have found the required block and read it down into a buffer as described in the preceeding section then the next three steps are to point at the first character to be overwritten, overwrite then Block Write back to the disk. Thus:-

```
PRINT#15,"B-P";2;8:REM START AT
CHARACTER 8

PRINT#2,AM$:REM OVERWRITE FROM
CHR 8 WITH AM$

PRINT#15,"B-W";2;0;T;S:REM WRITE THE
WHOLE LOT BACK TO DISK
```

7. THE 'NITTY GRITTIES'

INPUT#, like INPUT, can only 'bite off' chunks of data up to 80 characters long. It needs a 'delimiter' such as CHR$(13) or a comma to break up the data into such chunks and to signal the end of the last chunk. PRINT#ing can put a delimiter on the end of a chunk for you, but if the chunk is more than 80 characters long or if you try to input more chunks than you wrote, BASIC gets indigestion so badly it will hang up! So ensure that when you write the data, it is suitably subdivided, and that you read back in the same 'pattern' as you wrote to the disk.

When you do print to the disk, using PRINT#2,A$ for instance, as well as a CHR$(13), a CHR$(10), (line feed) also gets recorded on the disk, and will head the second and subsequent chunks of data you read back. Assuming you don't want this, the technique is to print using the following syntax:-

```
PRINT#2,A$;CHR$(13);
```

which will put a CHR$(13) onto the disk but suppress the normal CHR$(13) & CHR$(10), because of the final semi-colon.

Avoid null strings, that is to say consecutive CHR$(13)s, as part of your disk information. The best way is to test each string before you print it to the disk, using something along the following lines.

```
IFA$=""THENA$=CHR$(160):REM SHIFTED
SPACE

PRINT#2,A$;CHR$(13);
```

When a disk is VALIDATED, the Block Availability map, held on the diskette and used by the Block Allocate command to find the next free block, is cleaned up as part of the general tidy up that VALIDATE accomplishes. Since DOS didn't write the blocks in which you've recorded your direct access information, they are 'freed up' although the information in them remains intact. You can tackle the resultant problems by keeping your own availibility map and reallocating all the blocks. Since the index itself is such a map, you can use this with a utility routine to re-establish your claim on those blocks before DOS goes trampling all over them. Some people prefer to keep their direct access files on a seperate diskette well away from Sequential and Program files.

8. TO SUM UP

Disk syntax isn't as easy as BASIC, nor as forgiving. Mistakes in formatting information when writing to the disk often result in teeth-grinding hang-ups of BASIC when you read it back. But the steps are logical enough, and the syntax is not hard to grasp.

If you are going to use the same diskette as DOS is using for sequential and program files, then you must be prepared to dodge out of the way of DOS as it goes about its busy and useful purposes, almost blind to the activities of us direct access mice around its feet.

9. CONCLUSION

I hope this helps you with your disk operations. Once you've got the hang of this article your next move should be to re-read the Commodore Disk reference manual. Those who find this article too elementary for their applications might try using or adapting RANDOM 1, the example program in that manual.

Those of you who feel you need a little more assistance before you launch into major disk programming might like to book onto one of my DISK UTILISATION courses, where you will learn to write a program to handle the telephone index of the example in this article and many others besides.

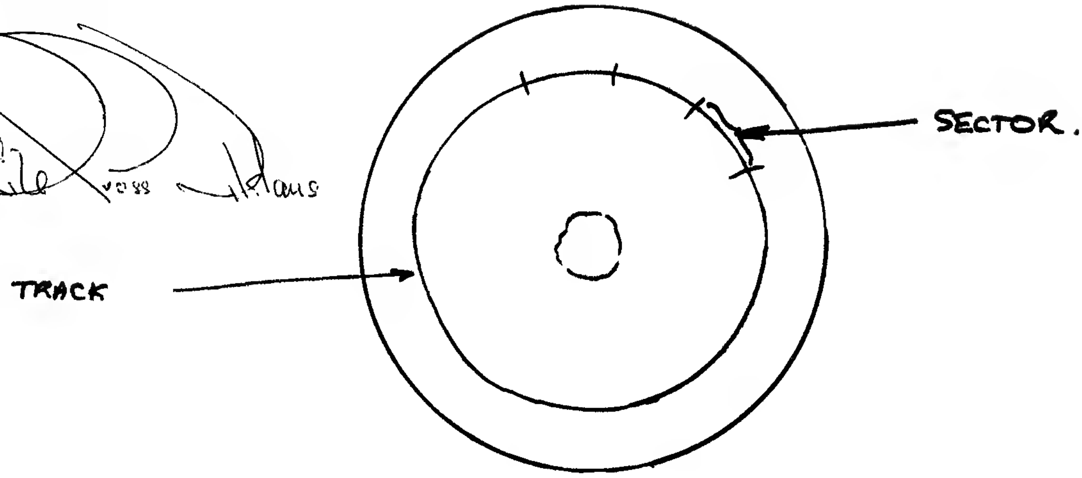


DIAGRAM 1.

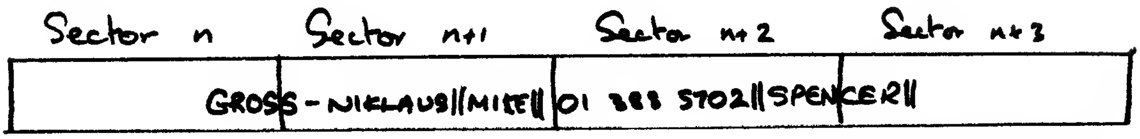


DIAGRAM 2. Concept of a sequential disk file.

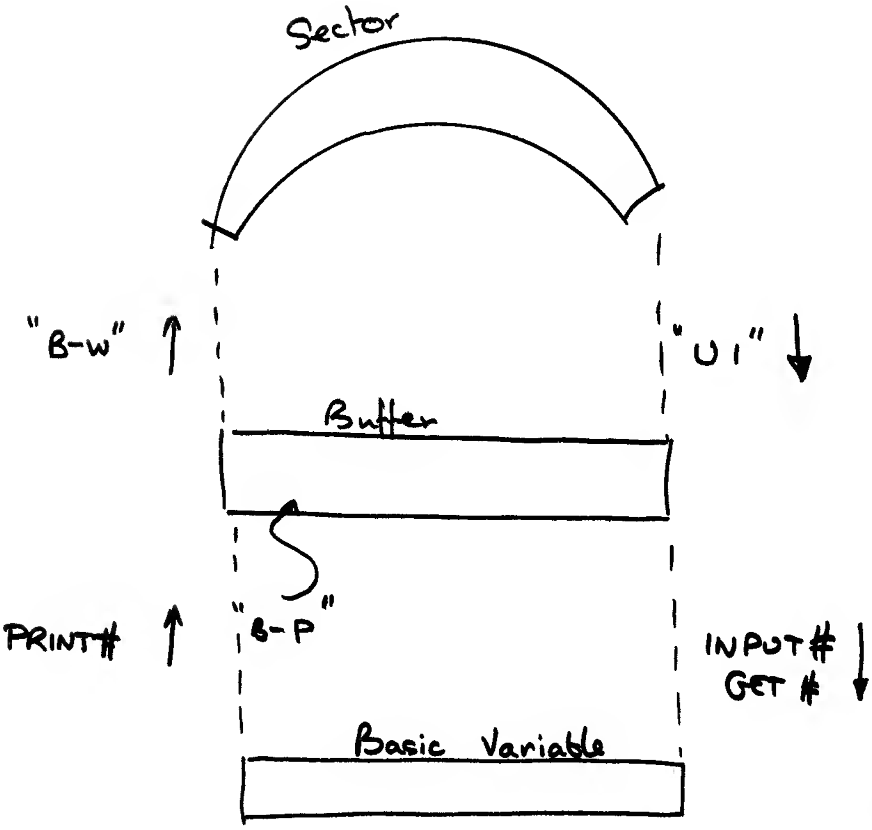


DIAGRAM 3. READY REFERENCE TO APPROPRIATE COMMANDS FOR DIRECT ACCESS.

Approved Product—Feature

COMPUTASTORE PAYROLL

Ethel Roberts can now enjoy her Sunday morning lie-in in bed - thanks to the Computastore PAYROLL program and Commodore PET.

For as Company Secretary to Smiths Books Ltd, a bustling bookselling and stationery business in Wigan, Lancs, Miss Roberts regularly found that she was so busy during the normal working week, that the payroll for the company's 48 employees was invariably left to the weekend.

Which generally meant that while the rest of her neighbours were still having a final forty winks, Ethel was spending her Sunday morning poring over the books, laboriously working out the employees' wages, tax and pension contributions by hand.

And with the prospect of having to take on an extra 20 staff in 1981 when the company expands into adjoining premises, Managing Director Trevor Smith decided that enough was enough; Smiths Books would computerise its payroll - and Ethel could have her lie-in.

"The trouble was that while Ethel is an extremely fast and efficient Company Secretary, she could usually do the payroll in under two hours, the shop is extremely busy right through the week and there just wasn't time for her to do the wages as well as all her other work", said Mr Smith.

"When we decided to acquire the property next door to make an even bigger shop, I realised that there was no way we could continue with hand-written ledgers considering the additional staff who would be required and the extra demands that would be placed on our accounting system".

"There was only one answer and that was to invest in a micro-computer. From then on, it was a comparatively simple step to choose the right machine and the right software package to handle our requirements".

The Computastore PAYROLL for the Commodore PET is an easy to use program designed for the first-time user and available from many Commodore official dealers (a list of dealers handling the program can be obtained by phoning 061 832 4761.)

The Computastore PAYROLL will accommodate most types of payroll, including piece work, hourly, weekly, and monthly calculations, and is so simple to use that accounts staff can be trained to operate it with the PET in just two hours.

From then on, wage calculations are fast, easy and accurate, and in the case of Smiths Books, now require less than 25 minutes to do the entire payroll for all 48 employees, including the production of printed payslips - a useful extra facility which the company did not have before.

Total cost of the installation, including PAYROLL? - around 2,500.

The software supplied by Computastore is intended for businesses with up to 275 employees and allows up to 10 gross pay elements (e.g. basic, overtime, shift allowance, bonus, etc), plus up to five after-tax adjustments, such as expenses, loans, union deductions and savings.

It will also calculate pension deductions as a percentage of the taxable gross, NI contributions for table letters A, B, C, D, E and C (contracted out), plus tax for basic and higher rates for all week 1 and month 1, NT, D prefix and all suffix tax codes.

A pre-printed payslip is produced in duplicate for each employee with totals and coin analysis at the end of each run. There is also a separate option for charities (with no employer's NI surcharge).

Employee details are stored on disk or cassette and the operator simply loads the progrm and stationery leaving the PET to process each employee record in sequence.

Gross pay, after-tax adjustments, tax, NI and pension (if applicable) are calculated down to net pay and employee details are then automatically written out to disk or tape.

Each package comes with comprehensive, step-by-step instructions and as an optional extra, Computastore will provide program updates for changes in tax or NI deductions.

The payslips meet all the requirements of the 1975 Employment Protection Act in that every deduction is itemised. Also, each time the system is used, a Log Report is produced to provide the user with an adequate audit trail.

The standard Disk Payroll costs 200 and cassette options 150, whilst tax and NI updates can be obtained for 15 a time. Pre-printed duplicate payslips to accompany the package cost 30 per thousand.

According to Smiths Books' Chief Accountant, Mr Neil Tarbuck, the company's investment in a Computastore PAYROLL package is already paying off and will prove even more cost-effective when new functions are added in the future.

"The program has proved excellent and now we know that the computer is capable of doing exactly what we require, I am evaluating other facilities it can provide, such as a purchase ledger system", he said.

All of which is good news for Commodore - and even better news for Ethel.

Beginning Machine Code

Paul Higginbottom

This series of articles form a guide to machine code programming on the PET, starting from first principles, and going on to explain how routines from the PET's own operating system can be used.

First, let's look at the heart of the microcomputer - the microprocessor.

A microprocessor relies on the fact that a voltage can be used to make a line into the processor 'high' or 'low'. In the case of the PET that means a line is either at 5 volts or it is grounded. These two states can be used to represent a '1' or a '0'. Or, if you like, 'ON' and 'OFF'. Since binary, or base 2 uses only the digits 0 and 1, the microprocessor has great significance in arithmetic applications.

Early programming was done by telling the microprocessor which lines to hold high and which to hold low. This is called micro-programming, and was very tedious. Soon, as expertise grew, a new level of programming became possible, where actual instructions could be used that were based on hardware architecture. For example, an instruction to enable a central store or register to be loaded with the present contents of a memory. Each memory location being defined as a specified number of ON's and OFF's like a binary number of a specified number of digits. Each 'ON' or 'OFF' digit is known as a 'BIT' and each actual memory location is called a 'BYTE' (a series of 8 'bit's).

Present day microprocessors have the basic capabilities of understanding an instruction set that can be used to manipulate data and move data around to different memory locations. This level of programming is called machine level programming (thus - machine code).

The operating system of the PET, (which includes the BASIC language, the screen editor, and the peripheral input/output routines etc.) is in fact, one huge machine code program that does all of the memory manipulation for you. It also evaluates all of the BASIC commands. The memory that it uses can be divided up into two catagories:

1) ROM (Read Only Memory)
2) RAM (Random Access Memory)

The ROM chips inside the Pet contain the routines that never need to change, and so are 'burnt' into the chips. If these routines weren't 'burnt' in, then as soon as the machine was switched off, all of the contents of the ROMs would be lost.

The RAM chips retain what ever bits have been set by the microprocessor and can thus be changed by it, only for as long as the computer system remains switched on. The microprocessor actually manages all the memory manipulation, so that, for example, when a line of BASIC is typed in it knows what area of memory to change.

Let's take a closer look at BASIC. When a BASIC program is run, each instruction is examined by the PET, and the corresponding piece of machine code in the operating system is executed. The operating system INTERPRETS them for the microprocessor. Thus, if Commodore chose to produce PETs with a different language all that would need changing is that part of the operating system that can interpret user programs.

Although microcomputers generally interpret and execute BASIC programs step by step, remembering things like where to jump back to after a subroutine has been finished, and where the start of the FOR NEXT loop is, this is not always the case. Often a COMPILER is used. This is a machine code program that actually turns the whole program into machine code! Anyway, let's just remember that the PET interprets BASIC, and executes the necessary routine(s) in the operating system.

If you examine a line of program written in BASIC; you will see that it consists of a line number, a statement with the necessary number of operands that instruction requires, and maybe another instruction, separated from the last by a colon. This is analogous to machine code, where a machine code instruction is stored as a number in a byte of memory, followed by the necessary number of operands (bytes of data) for that instruction. This represents either a number, or an address of a memory location inside the Pet that the program needs to access. The next machine code instruction follows on directly.

The trouble is that to look at it, machine code is just a stream of numbers stored in memory. Each of these numbers can be checked in turn using the PEEK instruction available from BASIC, or they can be inspected using the Machine Code Monitor (this utility is part of the ROM routines in the "new ROM" PETs, and is available as a program which can be loaded into RAM on the "old ROM" PETs.

Some way of representing machine code is needed to make it easier to use. This is done by using a three letter mnemonic associated with each instruction and a standard notation for the operand(s) associated with each instruction. So we need a program (BASIC or machine code) to enable us to create and edit a program in machine code, not as a series of numbers, but like BASIC for example, with reserved words for each instruction. Such a program is called an 'Assembler'. This allows the programmer to write and edit machine code programs in mnemonic format.

When the Pet is turned on, it automatically jumps into a machine code routine which checks free memory space it has, and then prints on the screen '### COMMODORE BASIC ###' etc. From then on, it expects you to type in a BASIC instruction, and if it

doesn't understand it, a 'SYNTAX ERROR?' is generated. So how do we get into machine code ? Well there are two BASIC instructions that provide a "bridge" from BASIC to machine code: One of them is 'SYS(X)', and the other is 'USR(X)'

The 'SYS' command requires a decimal location ('X'), and it will then start executing machine code from there. The 'USR' function has already had the address of where the machine code program starts, stored in two specific locations, and its paramater (X) is a number you can pass to the machine code program. This means that if, for example, a machine code program drew a square box in the left hand corner of the screen, then it is possible to pass the size of the box to the machine code program by making that the paramater put inside the USR command. For a fuller explanation of this command refer to the Pet User Manual supplied with your machine, or the "Pet Revealed" by Nick Hampshire which can be ordered through your local dealer.

So, to recap, machine code is a series of instructions followed by their respective operands, stored in a specific part of PET memory. To facilitate machine code programming some way of representing these instructions is needed, and so a complete set of mnemonics for all the instructions, and a standard notation has been devised. A special program is necessary to allow us to type and edit our programs using these mnemonics, which can be translated into the corresponding codes, and stored in memory. Such a program is called an Assembler.

A Disassembler reverses the process and allows an existing machine code program to be "translated back" into mnemonic notation. This is extremely useful when trying to decipher other people's machine code routines.

There are numerous assemblers and disassemblers on the market including a disk based assembler from Commodore. I use the Commodore assembler and find it to be about the best.

Having in this article examined the general features of machine code, I will in future articles look at the features of 6502 machine code. The 6502 is the micro-processor used inside the PET. In the next article I will look at the way PET's memory is organised and how a memory location is addressed.

Assembling an Assembler

R. J. Leman C. Eng. MIERE

Most owners of PETs will, at some time or other consider the use of machine code for solving problems, particularly if speed of operation is of importance.

Just how long it takes to produce a program depends on the ability of the programmer and the tools, both hardware and software, that he has available.

To show the speed advantage of machine code consider the following example. PET is controlling an experiment and periodically produces five results in the range 5 to 35 from an instrument and logs them into memory. So that the user can keep an eye on progress a simple horizontal bar chart is required, but PET must be "off line" to the experiment for as short a time as possible. This program shows a BASIC solution to the problem, and sets up a machine code equivalent in the second cassette buffer.

```
 10 S=32768:T=TI
 20 FOR X = 5 TO 0 STEP-1
 30 A = PEEK (870+X)
 40 FOR Y=ITOA
 50 POKE S+Y,102
 60 NEXT Y
 70 S=S+40
 80 NEXT X:PRINT TI-T"JIFFIES"
190 AD = 826
200 READ Q$
210 IF Q$ = "*"THEN STOP
220 POKE AD, VAL (Q$)
230 AD=AD+1
240 GOTO 200
250 DATA 169,0,141,76,3,169,128,141,77,
    3,160,5,190,102,3,169,102,157,200,
    128,202
260 DATA 208,250,136,48,17,173,76,3,24,
    105,40,141,76,3,144,231,238,77,3,
    76,70,3
270 DATA 96,10,15,20,25,30,35,*
```

Type RUN and see the BASIC program draw the bar chart. Now clear the screen, move the cursor down several places and type SYS826. The speed advantage of machine code is quite apparent.

The ability to write machine code is a skill that all PET programmers should aquire, particularly as the SYS and USR commands allow machine code to be mixed with BASIC to produce optimum results. Getting started in BASIC is not too difficult, particularly with the wealth of good books on the subject, but making the first steps in machine code can result in a rapid membership application to Alcholics Anonymous! The lack of friendly ERROR messages and frustrating hang ups can daunt the most enthusiastic programmer. Obviously a good Assembler package helps with progress, and "hereby hangs the tale".

Assemblers vary in cost, complexity and convenience of use. Particularly inconvenient to the beginner are Assembly Systems requiring repeated loading of Editor, Assembler, debugging package and the necessary files. As an illustration, the time taken to produce the machine code for the example, using a multi-program system, including the correction of a mistyped mnemonic found at the Assembly stage was 56 minutes, of which 30 were spent watching the cassettes go round!!

This frustrating delay resulted in the production of ASMPAC8 and it's related aids.

In a nutshell, ASMPAC8 had to run in my antique 8K PET without any additional memory or provide adequate facilities for entering and editing Assembly Language Text, Assembling into memory or object code file and disassemble programs located in free memory. This was to be done without the use of files other than as a precaution against crash situations or for record purposes. Obviously a limit on the number of Assembly Language statements had to be accepted, but the ability to produce free standing Object-code in a form suitable for direct saving was essential. Reliability had to be high, the system easy to use and had to adhere to as many of the standard Assembly conventions detailed in the MOS 6502 Programming Manual, as possible.

After much developement ASMPAC8 was produced and satisfied all of the above requirements. Fifty lines of Assembly Text could be handled without the use of files, and larger requirements dealt with in sections or by the addition of more memory.

Two short programs were produced to operate on the object code files produced by ASMPAC8. AUTOSTRINGS was produced to allow short routines to be stored as coded BASIC strings, and this satisfied the need for a convenient way to include short routines in predominantly BASIC programs.

The obvious place for such code is in the second cassette buffer, but optionally the top 256 bytes may be reserved and used instead. AUTOSTRINGS does this automatically using information from the Object code file.

AUTOLOAD8 accepts Object code files from ASMPAC8 and produces a free standing program that is locatable in the first 2000 bytes of the BASIC RAM area. Such programs may be SAVED, VERIFIED, LOADED and RUN, just like BASIC but with the enhanced speed that machine code offers.

Both support programs automatically discard their file reading sections once they have finished with them, leaving a tidy program.

Once the programs were in a reasonably polished form, they were tested by the simple technique of use, more use and yet more use. Minor additions such as the ability to display the Label table and check free memory were incorporated. ASMPAC8 is now ready, having met all of

it's target requirements, to help others learn how to produce and take advantage of the machine code capabilities of PET. One word of warning - it's ADDICTIVE.

ASMPAC8 and its sister programs are approved by Commodore - full details from: JCL Software, 47 London Road, Southborough TUNBRIDGE WELLS, Kent.

Programming

DISK BUG

If you write variable length sequential files and read back with INPUT# and test for End Of File with ST, beware! Files 254 bytes long or any integer multiple thereof will cause PET to hang up if read by INPUT# and a status flag test. The cause of the bug is an extra carriage return being placed at the end of a file if it is a factor of 254 bytes long. The solution is to write a character without a carriage return (CHR$(13)) as the final character in the file, a line feed (CHR$(10)) is suggested. (INPUT# will still accept the character even when a carriage return is not at the end of file).

The program to illustrate the problem of the fix is shown below:
Congratulations to Nick Marcopoulous for an elegant solution.

```
 4 REM RUN THIS PROGRAM WITH AND WITHOUT LINE90
 5 FORX=1TO3:PRINT
 10 OPEN1,8,8,"@0:TEST,S,W"
 30 FORJ=1TOX
 40 PRINT#1,"X";
 50 NEXT
 60 FORI=1TO126
 70 PRINT#1,"X";CHR$(13);
 80 NEXT
 90 PRINT#1,CHR$(10);
 100 CLOSE1
 110 OPEN1,8,5,"0:TEST,S"
 120 INPUT#1,A$:SS=ST
 130 PRINTA$;
 140 IFSSAND64THENCLOSE1:NEXT:END
 150 GOTO120
READY.
```

Tony Winter the author of the Commodore Approved "Business Efficiency" packages, was first to bring the DISKBUG to our attention and to suggest a solution - using a byte count method. Tony has two other tips for Disk Users:-

1. The cause of DOS Rename failures is still obscure but can be dealt with by simply checking the error number through channel 15, immediately after a Rename attempt and if it turns out to be 62 then try again say three times; if all attempts fail do a copy function. One might prefer to always Copy, but for longer files there will be a significant delay in processing, so the Rename attempt will serve as a time saver if it works.

2. Unreliable data transfer (data crossing over to wrong files due to upset block availability map) is best compensated for by all file changes being followed by a Verify on the data disk.

BUG IN RANDOM 1.0

Users of the program RANDOM 1.0 which is a collection of BASIC sub-routines to manage random access files may have noticed that in certain circumstances the syntax error in line 820 is encountered. This is of course fatal and causes the program to terminate. After hours of research in our Software laboratory, we have come to the conclusion that line 820 should contain a reference to the variable EN. The line should read:

```
820 E=0: IFT=0 THEN EN=40:GOTO 1900
```

This is an obvious bug but has only been reported twice to us. Dealers should update existing demonstration diskettes on behalf of their users.

PRINTER FIX

Some users have reported problems on the printer when changing format. The following is taken from the Canadian Pet Newsletter "The Transactor"

There has been a bug detected with the formatting feature of the 2022 and 2023 Printers but fortunately, Kim Lantz of North Sydney, Nova Scotia, has found the fix.

It seemed that setting up the first format was no problem, but changing to a second format was. When PRINTing to the printer, the last character to be sent to a line is a CRLF. This is done for obvious reasons but, the Carriage Return is printed on the current line and the Line Feed is printed on the next line. The Line Feed character is of course not printed on the paper but the printer "sees" it as the first character of the new line and when the printer is anywhere but the absolute beginning of a line, it doesn't like changing the format.

Therefore, anything that is output to secondary address 1 of the printer should be followed by...

;CHR$(13);

```
e.g.  OPEN 1,1,1
      PRINT #1, X;CHR$(13);
      PRINT #1, "PET";CHR$(13);
```

...especially when the format string is about to be changed. This is also true for secondary address 0.

The above can of course be shortened by first equating R$ to CHR$(13) and using R$ in place of CHR$(13). Also, the first semi-colon is not necessary when preceeded by a closing quote or another string variable but is necessary when following numeric variables.

However, the general idea is to keep the printer in the 0'th position after a carriage return when the format string is to be changed.

Another solution has been independently arrived at by Henri Rubin our South African dealer and is listed below:-

```
10 A$="COMMODORE":A=123:B=456
20 F$(1)="AAAAAAAAA   999 999"
30 F$(2)="  AAAAAAAAA   999   999"
40 F$(3)="       AAAAAAAAA     999   999"
50 OPEN1,4,2:OPEN2,4,1
80 FORI=1TO3
85 PRINT#1,F$(I)
90 PRINT#2,A$CHR$(29);A;B;
100 PRINT#1,"":PRINT#2,""
110 NEXT
READY.
```

Supermon 2

In CPUCN Volume 2 Issue 3 we published the instruction program for Supermon and also Jim Butterfield's re-location routine which will assist you in running the program and saving it.

In this issue we print the Hex listing for Supermon itself. Our apologies for not making it clear that this was to follow. Supermon adds a simple assembler and disassembler to the built-in machine code monitor in "level 2" ROM PETs, also the ability to single step a machine code program, to hunt memory and a number of other very useful utilities.

Supermon is ideal as a teaching tool for the novice machine code programmer and previews many of the advanced features available in Extramon which is part of the Commodore assembler developement system.

```
.:  06B0 4A 3A 8E 00 00 00 AD FF
.:  06B8 FE 00 85 34 AD FF FF 00
.:  06C0 85 35 AD FF FC 00 8D FA
.:  06C8 03 AD FF FD 00 8D FB 03
.:  06D0 00 00 00 A2 08 DD FF DE
.:  06D8 00 D0 0E 86 B4 8A 0A AA
.:  06E0 BD FF E9 00 48 BD FF AD
.:  06E8 FF FE 00 85 34 AD FF FF
.:  06F0 00 85 35 AD FF FC 00 8D
.:  06F8 FA 03 AD FF FD 00 8D FB
.:  0700 03 00 00 00 A2 08 DD FF
.:  0708 DE 00 D0 0E 86 B4 8A 0A
.:  0710 AA BD FF E9 00 48 BD FF
.:  0718 E8 00 48 60 CA 10 EA 4C
.:  0720 F7 E7 A2 02 2C A2 00 00
.:  0728 00 B4 FB D0 08 B4 FC D0
.:  0730 02 E6 DE D6 FC D6 FB 60
.:  0738 20 EB E7 C9 20 F0 F9 60
.:  0740 A9 00 00 00 8D 00 00 00
.:  0748 01 20 FA 8C 00 20 BE E7
.:  0750 20 AA E7 90 09 60 20 EB
.:  0758 E7 20 A7 E7 B0 DE 4C F7
.:  0760 E7 20 CD FD CA D0 FA 60
.:  0768 E6 FD D0 02 E6 FE 60 A2
.:  0770 02 B5 FA 48 BD 0A 02 95
.:  0778 FA 68 9D 0A 02 CA D0 F1
.:  0780 60 AD 0B 02 AC 0C 02 4C
.:  0788 FA DD 00 A5 FD A4 FE 38
.:  0790 E5 FB 85 CF 98 E5 FC A8
.:  0798 05 CF 60 20 FA 94 00 20
.:  07A0 97 E7 20 FA A5 00 20 FA
.:  07A8 BE 00 20 FA A5 00 20 FA
.:  07B0 D9 00 20 97 E7 90 15 A6
.:  07B8 DE D0 64 20 FA D0 00 90
.:  07C0 5F A1 FB 81 FD 20 FA B7
.:  07C8 00 20 D5 FD D0 EB 20 FA
.:  07D0 D0 00 18 A5 CF 65 FD 85
.:  07D8 FD 98 65 FE 85 FE 20 FA
.:  07E0 BE 00 A6 DE D0 3D A1 FB
.:  07E8 81 FD 20 FA D0 00 B0 34
.:  07F0 20 FA 78 00 20 FA 7B 00
.:  07F8 4C FB 27 00 20 FA 94 00
.:  0800 20 97 E7 20 FA A5 00 20
.:  0808 97 E7 20 EB E7 20 B6 E7
.:  0810 90 14 85 B5 A6 DE D0 11
.:  0818 20 FA D9 00 90 0C A5 B5
.:  0820 81 FB 20 D5 FD D0 EE 4C
.:  0828 F7 E7 4C 56 FD 20 FA 94
.:  0830 00 20 97 E7 20 FA A5 00
.:  0838 20 97 E7 20 EB E7 A2 00
.:  0840 00 00 20 EB E7 C9 27 D0
.:  0848 14 20 EB E7 9D 10 02 E8
.:  0850 20 CF FF C9 0D F0 22 E0
.:  0858 20 D0 F1 F0 1C 8E 00 00
.:  0860 00 01 20 BE E7 90 C6 9D
.:  0868 10 02 E8 20 CF FF C9 0D
.:  0870 F0 09 20 B6 E7 90 B6 E0
.:  0878 20 D0 EC 86 B4 20 D0 FD
.:  0880 A2 00 00 00 A0 00 00 00
.:  0888 B1 FB DD 10 02 D0 0C C8
.:  0890 E8 E4 B4 D0 F3 20 6A E7
.:  0898 20 CD FD 20 D5 FD A6 DE
.:  08A0 D0 92 20 FA D9 00 B0 DD
.:  08A8 4C 56 FD 20 FA 94 00 8D
.:  08B0 0D 02 A5 FC 8D 0E 02 A9
.:  08B8 04 A2 00 00 00 85 B8 86
.:  08C0 B9 A9 93 20 D2 FF A9 16
.:  08C8 85 B5 20 FC 10 00 20 FC
.:  08D0 6D 00 85 FB 84 FC C6 B5
.:  08D8 D0 F2 A9 91 20 D2 FF 4C
.:  08E0 56 FD A0 2C 20 15 FE 20
.:  08E8 6A E7 20 CD FD A2 00 00
.:  08F0 00 A1 FB 20 FC 7C 00 48
.:  08F8 20 FC C2 00 68 20 FC D8
.:  0900 00 A2 06 E0 03 D0 12 A4
.:  0908 B6 F0 0E A5 FF C9 E8 B1
.:  0910 FB B0 1C 20 FC 65 00 88
.:  0918 D0 F2 06 FF 90 0E BD FF
.:  0920 4A 00 20 FD 4D 00 BD FF
.:  0928 50 00 F0 03 20 FD 4D 00
.:  0930 CA D0 D5 60 20 FC 70 00
.:  0938 AA E8 D0 01 C8 98 20 FC
.:  0940 65 00 8A 86 B4 20 75 E7
.:  0948 A6 B4 60 A5 B6 38 A4 FC
.:  0950 AA 10 01 88 65 FB 90 01
.:  0958 C8 60 A8 4A 90 0B 4A B0
.:  0960 17 C9 22 F0 13 29 07 09
.:  0968 80 4A AA BD FE F9 00 B0
.:  0970 04 4A 4A 4A 4A 29 0F D0
```

```
.:  0978 04 A0 80 A9 00 00 00 AA
.:  0980 BD FF 3D 00 85 FF 29 03
.:  0988 85 B6 98 29 8F AA 98 A0
.:  0990 03 E0 8A F0 0B 4A 90 08
.:  0998 4A 4A 09 20 88 D0 FA C8
.:  09A0 88 D0 F2 60 B1 FB 20 FC
.:  09A8 65 00 A2 01 20 FA B0 00
.:  09B0 C4 B6 C8 90 F1 A2 03 C4
.:  09B8 B8 90 F2 60 A8 B9 FF 57
.:  09C0 00 8D 0B 02 B9 FF 97 00
.:  09C8 8D 0C 02 A9 00 00 00 A0
.:  09D0 05 0E 0C 02 2E 0B 02 2A
.:  09D8 88 D0 F6 69 3F 20 D2 FF
.:  09E0 CA D0 EA 4C CD FD 20 FA
.:  09E8 94 00 20 D5 FD 20 D5 FD
.:  09F0 20 97 E7 20 FA A5 00 20
.:  09F8 97 E7 20 CA FD 20 FA D9
.:  0A00 00 90 09 98 D0 13 A5 CF
.:  0A08 30 0F 10 07 C8 D0 0A A5
.:  0A10 CF 10 06 20 75 E7 4C 56
.:  0A18 FD 4C F7 E7 20 FA 94 00
.:  0A20 A9 03 85 B5 20 EB E7 20
.:  0A28 A7 FD D0 F8 AD 0D 02 85
.:  0A30 FB AD 0E 02 85 FC 4C FB
.:  0A38 F1 00 C5 B9 F0 03 20 D2
.:  0A40 FF 60 A9 03 A2 24 85 B8
.:  0A48 86 B9 20 D0 FD 78 AD FF
.:  0A50 FA 00 85 90 AD FF FB 00
.:  0A58 85 91 A9 A0 8D 4E E8 CE
.:  0A60 13 E8 A9 2E 8D 48 E8 A9
.:  0A68 00 00 00 8D 49 E8 AE 06
.:  0A70 02 9A 4C F1 FE 20 7B FC
.:  0A78 68 8D 05 02 68 8D 04 02
.:  0A80 68 8D 03 02 68 8D 02 02
.:  0A88 68 8D 01 02 68 8D 00 00
.:  0A90 00 02 BA 8E 06 02 58 20
.:  0A98 D0 FD 20 BF FD 85 B5 A0
.:  0AA0 00 00 00 20 9A FD 20 CD
.:  0AA8 FD AD 00 00 00 02 85 FC
.:  0AB0 AD 01 02 85 FB 20 6A E7
.:  0AB8 20 FC 18 00 20 01 F3 C9
.:  0AC0 F7 F0 F9 20 01 F3 D0 03
.:  0AC8 4C 56 FD C9 FF F0 F4 4C
.:  0AD0 FD 60 00 00 00 00 20 FA
.:  0AD8 94 00 20 97 E7 8E 11 02
.:  0AE0 A2 03 20 FA 8C 00 48 CA
.:  0AE8 D0 F9 A2 03 68 38 E9 3F
.:  0AF0 A0 05 4A 6E 11 02 6E 10
.:  0AF8 02 88 D0 F6 CA D0 ED A2
.:  0B00 02 20 CF FF C9 0D F0 1E
.:  0B08 C9 20 F0 F5 20 FE F0 00
.:  0B10 B0 0F 20 CB E7 A4 FB 84
.:  0B18 FC 85 FB A9 30 9D 10 02
.:  0B20 E8 9D 10 02 E8 D0 DB 8E
.:  0B28 0B 02 A2 00 00 00 86 DE
.:  0B30 A2 00 00 00 86 B5 A5 DE
.:  0B38 20 FC 7C 00 A6 FF 8E 0C
.:  0B40 02 AA BD FF 97 00 20 FE
.:  0B48 D5 00 BD FF 57 00 20 FE
.:  0B50 D5 00 A2 06 E0 03 D0 12
.:  0B58 A4 B6 F0 0E A5 FF C9 E8
.:  0B60 A9 30 B0 1D 20 FE D2 00
.:  0B68 88 D0 F2 06 FF 90 0E BD
.:  0B70 FF 4A 00 20 FE D5 00 BD
.:  0B78 FF 50 00 F0 03 20 FE D5
.:  0B80 00 CA D0 D5 F0 06 20 FE
.:  0B88 D2 00 20 FE D2 00 AD 0B
.:  0B90 02 C5 B5 D0 59 20 97 E7
.:  0B98 A4 B6 F0 2B AD 0C 02 C9
.:  0BA0 9D D0 1C 20 FA D9 00 90
.:  0BA8 09 98 D0 4A A6 CF 30 46
.:  0BB0 10 07 C8 D0 41 A6 CF 10
.:  0BB8 3D CA CA 8A A4 B6 D0 03
.:  0BC0 B9 FC 00 00 00 91 FB 88
.:  0BC8 D0 F8 A5 DE 91 FB 20 FC
.:  0BD0 6D 00 85 FB 84 FC A0 41
.:  0BD8 20 15 FE 20 6A E7 20 CD
.:  0BE0 FD 4C FD DE 00 20 FE D5
.:  0BE8 00 86 B4 A6 B5 DD 10 02
.:  0BF0 F0 0C 68 68 E6 DE F0 03
.:  0BF8 4C FE 30 00 4C F7 E7 E8
.:  0C00 86 B5 A6 B4 60 C9 30 90
.:  0C08 03 C9 47 60 38 60 40 02
.:  0C10 45 03 D0 08 40 09 30 22
.:  0C18 45 33 D0 08 40 09 40 02
.:  0C20 45 33 D0 08 40 09 40 02
.:  0C28 45 B3 D0 08 40 09 00 00
.:  0C30 00 22 44 33 D0 8C 44 00
.:  0C38 00 00 11 22 44 33 D0 8C
.:  0C40 44 9A 10 22 44 33 D0 08
.:  0C48 40 09 10 22 44 33 D0 08
.:  0C50 40 09 62 13 78 A9 00 00
.:  0C58 00 21 81 82 00 00 00 00
.:  0C60 00 00 59 4D 91 92 86 4A
.:  0C68 85 9D 2C 29 2C 23 28 24
.:  0C70 59 00 00 00 58 24 24 00
.:  0C78 00 00 1C 8A 1C 23 5D 8B
.:  0C80 1B A1 9D 8A 1D 23 9D 8B
.:  0C88 1D A1 00 00 00 29 19 AE
.:  0C90 69 A8 19 23 24 53 1B 23
.:  0C98 24 53 19 A1 00 00 00 1A
.:  0CA0 5B 5B A5 69 24 24 AE AE
.:  0CA8 A8 AD 29 00 00 00 7C 00
.:  0CB0 00 00 15 9C 6D 9C A5 69
.:  0CB8 29 53 84 13 34 11 A5 69
.:  0CC0 23 A0 D8 62 5A 48 26 62
.:  0CC8 94 88 54 44 C8 54 68 44
.:  0CD0 E8 94 00 00 00 B4 08 84
.:  0CD8 74 B4 28 6E 74 F4 CC 4A
.:  0CE0 72 F2 A4 8A 00 00 00 AA
.:  0CE8 A2 A2 74 74 74 72 44 68
.:  0CF0 B2 32 B2 00 00 00 22 00
.:  0CF8 00 00 1A 1A 26 26 72 72
.:  0D00 88 C8 C4 CA 26 48 44 44
.:  0D08 A2 C8 04 22 10 20 2D 2F
.:  0D10 33 54 46 48 44 43 2C 41
.:  0D18 49 4E 00 00 00 FA E8 00
.:  0D20 FB 3C 00 FB 6A 00 FB DD
.:  0D28 00 FC FD 00 FD 30 00 FD
.:  0D30 DA 00 FD 54 00 55 FD FD
.:  0D38 84 00 FA 5D 00 FA 46 00
.:  0D40 4C 49 00 00 00 00 00 AA
```

Hardware Reset

Jim Butterfield recently demonstrated his recomended method for un-crashing a "level 2" ROM PET. This method has the supreme advantage of leaving any BASIC or machine code programs intact, i.e. the method sucessfully by-passes PET's inclination of destructively testing RAM as part of its "cold start" routine.

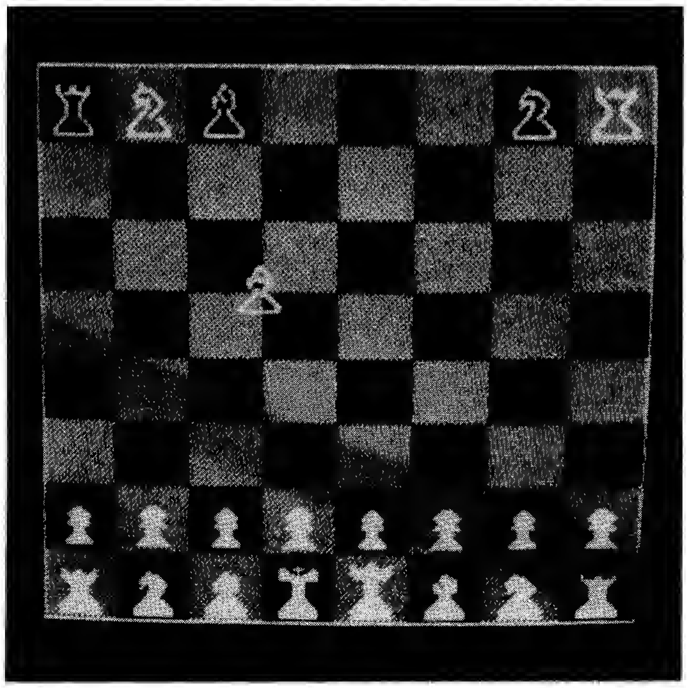
Pin 5 (Diagnostic Sense) of the user port must be grounded, while J4 22 (Reset) of the memory expansion bus is grounded momentarily. The ground to Diagnostic Sense can then be released. PET should come up with the machine code monitor. If you wish to return to basic then type "X" followed by RETURN, then "CLR" followed by

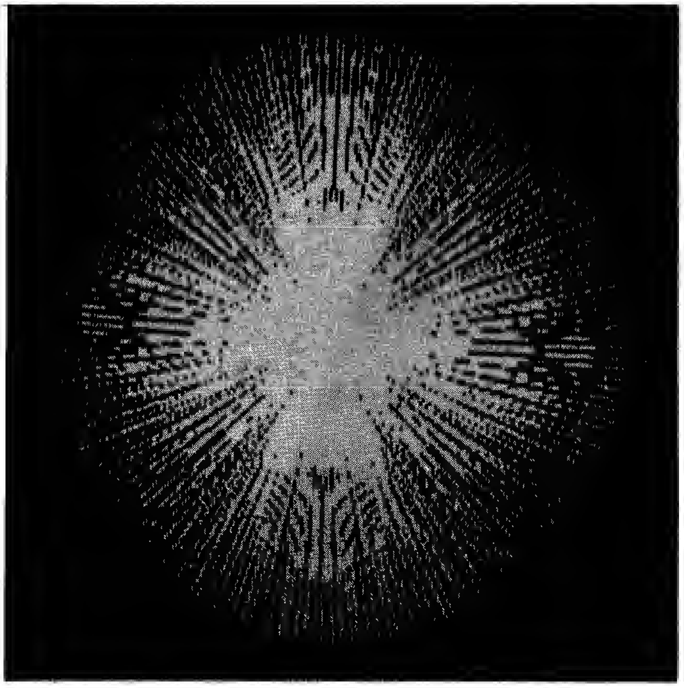
RETURN. Your program can then be run provided tht the house-keeping variables in Page 0 RAM have not been tangled up by the crash.

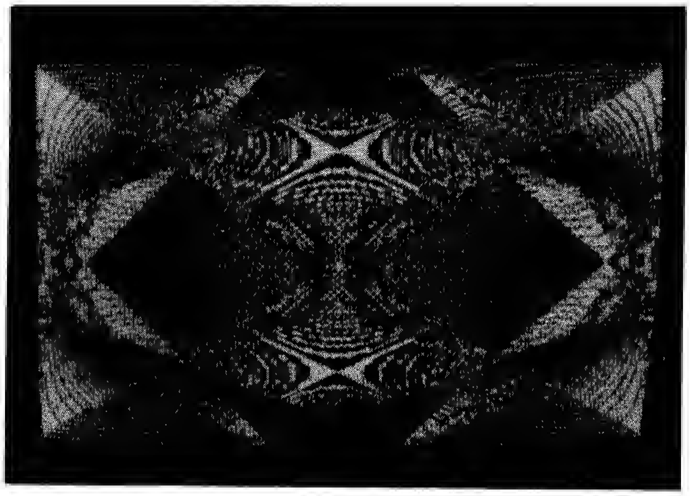
If you wish to stay in the machine code monitor, type a semicolon followed by RETURN then place the cursor over the stack pointer which will have become "01" and type over with "FA" followed by RETURN. You can now continue to work with the machine code monitor.

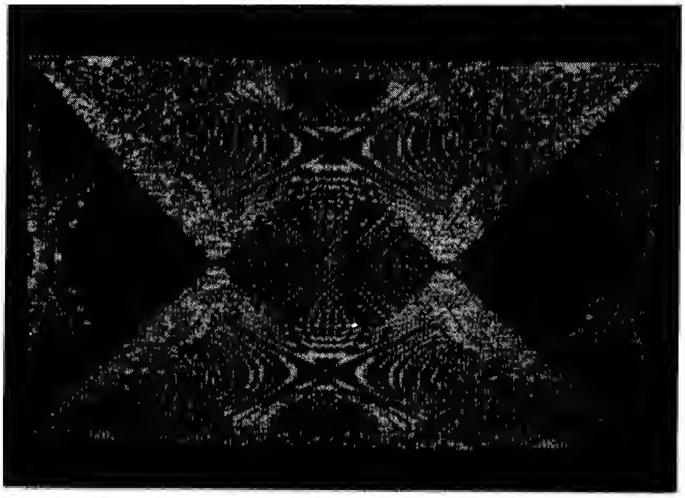
Bits and Bytes

CONTINUOUS GRAPHICS ON THE PET!









IJJ Design Ltd, the Wiltshire based importers of the complete range of MTU boards for the PET and KIM, demonstrated their continuous graphics board at a recent dealer exhibition.

Some special utility software has been written for the system by UK machine code programmer Pete Dowson, and some stunning demonstrations were shown of animated chess pieces and superb graphics.

IJJ have kindly lent us a set of boards for review and an article will appear soon in CPUCN. Meanwhile, to whet the reader's appetites, we have published some photographs showing the capabilities of the new boards.

PETS AT THE M.E. CENTRE

The M.E. Centre is the new National Microprocessor and Electronics Centre situated in the London World Trade Centre.

It is a permanent exhibition of all aspects of electronics with free admission 5 days a week.

Commodore have a stand on which we are exhibiting an 8K Pet and a 32K PET with Floppy Disk (the M.E. Centre also have their own PET configuration which is used for visitor registration and mailing shots).

A brochure giving further details of the M.E. Centre and exhibitions is enclosed. If you want hands-on experience with a 32K PET, Disk or even test a program, then, subject to availability (and other User Club members!) the M.E. Centre is the place to go.

FORTH LANGUAGE FOR THE PET

Petsoft have introduced a modestly priced interactive FORTH Compiler/Interpreter to run on the PET.

FORTH is a unique high level language, which has recently become popular in the United States, where it is widely used in astronomical and general scientific applications.

The principal advantages of the PET FORTH are that it is extremely fast and requires only a small amount of memory. The language lends itself naturally to structured programming.

Petsoft's PET FORTH package contains a dictionary of 200 'words', each of which approximates to a subroutine in BASIC. Being vocabulary based the user can tailor the system to resemble the needs and structures of any specific application. Also included are a built-in incremental assembler and text editor.

PET-FORTH is available at 30 + VAT from Commodore official dealers or direct from Petsoft.

NEW PACKAGE FOR NUMERICAL CONTROL

Taylor Wilson Systems Ltd, have recently released "TAPEPREP", a complete hardware and software micro-computer package for under £4000, for the preparation of control tapes for N.C. (Numerically Controlled) Machine Tools.

Until today the options available to produce the control tapes and store routines have either been tedious and time consuming or relatively easy but expensive, requiring the use of costly computer systems.

"TAPEPREP" is a cost effective solution to the problem of the production of control tapes to keep N.C. tools cutting metal rather than standing idle whilst the next tape is prepared. The all in price of just under £4000 includes the hardware, set of programs, instruction manuals, dust covers and initial supply of floppy disks.

APOLOGIES

Our apologies to Paul Higginbottom for mis-spelling his name in the last CPUCN. Also a mistake crept in to the one line routine to convert PEEK and POKE codes to ASCII. This should have been:

```
B=(AAND127)OR((AAND64)*2)OR((64-AAND32)*2)
```

This routine was, however, correctly reproduced in the sample program.

PET DESK

Tom Allen of Allen Computers has produced a smart teak faced desk which will accomodate a Commodore PET, up to 3 Floppy Disk Units and Printer. Size 61 x 31 inches. One off price is £385 excluding carriage and VAT. Full details from:

Allen Computers
16 Hainton Avenue
Grimsby
South Humberside

PET 2516/2716 EPROM PROGRAMMER

The G. R. Electronics EPROM programmer package turns the Commodore PET into a convenient and flexible EPROM programmer. As an added advantage it permits a user's machine code routine to be written in firmware to be run on the PET itself, increasing its power and speed of operation.

The package comprises a fully assembled and tested plug-in PCB and software cassette which allow the PET to read from and write to type 2516/2716 EPROMs. It is suitable for both the 2001 type PET with a minimum 8K of RAM and the 3016 large keyboard version. NOTE: the type of PET should be specified when ordering as the software differs.

The software routines make use of the PET's keyboard and screen to provide display and editing facilities not available on conventional EPROM programmers. This gives considerably greater flexibility in writing and modifying software to be written into EPROM.

Full details from:-

G R Electronics Ltd
Fairoak House
Church Road
Newport
Gwent
NPT 7EJ

Tel: 0633 214147/8

JOB EVALUATION PACKAGE FOR THE PET

Petsoft have released a Job Evaluation package, selling for just £25 to run on the PET.

Based on a method devised by management consultants for use on mainframe computers, Job Evaluation determines the best formula for evaluating jobs based on the thinking of the Company concerned. The number of factors contributing to job value are entered (for example) education necessary, responsibility over other personnel, working conditions, etc.), along with the names of several benchmark jobs, such as: doctor, policeman, secretary, etc.

The computer then asks a large number of 'paired comparison' questions in the form "Who has better working hours, firemen or a retail salesman?". Replies are recorded by means of a unique answer-entry system, for later analysis.

The program uses multiple regression analysis to produce the final Job Evaluation Formula - with all the component job factors having the correct weighting. This formula can be used to provide a relative valuation for almost any job in the company, and hence can be used to establish pay structures and grades.

Job Evaluation will not instantly solve all industrial relations problems, but will enable a company to evaluate jobs in a more objective and factual manner. Petsoft are aiming the program at managers of large firms, personnel departments and management consultants.

Priced at £25 + VAT, this program is available through most Commodore dealers.

PET in Education

CALLING ALL PET USERS IN EDUCATION

Nick Green

If you are a teacher using the PET, or a representative of a group of teachers working with PET, please contact us so that we may compile an Education Users Directory in order to encourage more efficient sharing of resources. In particular we are interested in Secondary and Primary schools, but Universities and Colleges who are doing work relevant to these schools are also of interest.

We are hoping to discover what it is that can be done to make the PET even more of a success in schools than it has been up to now. We don't know if any of you read the recent issue of "Educational Computing" but the praise that was heaped on our dearly loved PET was quite embarrassing! It is, of course, apparent that we need a lot more software with appropriate teacher notes where necessary and some special hardware, perhaps.

In the next few months we will be preparing a new "PET in Education" brochure, and it is our intention to give publicity to the various teachers' groups that have formed around the PET, both within existing micro computer societies and outside. E.g. PET Education Group set up by Chris Smith (Department of Physiology, Queen Elizabeth College, London; PET group within MUSE. PET software has been prepared by Bob Lewis at Chelsea College and Rosemary Fraser of ITMA in Plymouth.

Some of the work of these people will be known to you and some may not be. We, of course, have a small band of teachers who have been sending us material for more than a year now. It will be my aim, where possible, to give more publicity to those centres where activity is greatest and where developement of software is feasible and from what sources hardware of use in the classroom might be obtained.

This information will be published in a resource directory which I hope will greatly simplify the problems of the teacher in the classroom who wishes to introduce his students to micro computing.

No doubt there are many of you with your own ideas about the priorities and what particular developements you would like to see. Please write to me at Commodore Business Machines, 818 Leigh Road, Slough (Telephone Slough 74111).